contact时候 产生ER
1.前后ER比较检测出packet dropping
2.利用ER的一致性来保证 不会被伪造misreporting 识别forge-buffer forge-inactive forge-transcation
3.记录频繁制造相互传输记录的collude gauge_list
4.评价方法 ρ δ 乘 加 参数

Mitigating Routing Misbehavior in Disruption Tolerant Networks

Qinghua Li, *Student Member, IEEE*, and Guohong Cao, *Fellow, IEEE*

***Abstract*—In disruption tolerant networks (DTNs), selfish or malicious nodes may drop received packets. Such routing misbehavior reduces the packet delivery ratio and wastes system resources such as power and bandwidth. Although techniques have been proposed to mitigate routing misbehavior in mobile ad hoc networks, they cannot be directly applied to DTNs because of the intermittent connectivity between nodes. To address the problem, we propose a distributed scheme to detect packet dropping in DTNs. In our scheme, a node is required to keep a few signed contact records of its previous contacts, based on which the next contacted node can detect if the node has dropped any packet. Since misbehaving nodes may misreport their contact records to avoid being detected, a small part of each contact record is disseminated to a certain number of witness nodes, which can collect appropriate contact records and detect the misbehaving nodes. We also propose a scheme to mitigate routing misbehavior by limiting the number of packets forwarded to the misbehaving nodes. Trace-driven simulations show that our solutions are efficient and can effectively mitigate routing misbehavior.**

***Index Terms*—Detection, disruption tolerant networks, mitigation, routing misbehavior, security.**

I. Introduction

DISRUPTION tolerant networks (DTNs) [1]–[3] exploit the intermittent connectivity between mobile nodes to transfer data. Due to a lack of consistent connectivity, two nodes exchange data only when they move into the transmission range of each other (which is called a *contact* between them). Thus, DTN routing usually follows "store-carry-forward;" i.e., when a node receives some packets, it stores these packets in its buffer, carries them around until it contacts another node, and then forwards the packet.

In DTNs, a node may misbehave by dropping packets even when it has sufficient buffers. Routing misbehavior can be caused by selfish nodes that are unwilling to spend resources such as power and buffer on forwarding packets of others, or caused by malicious nodes that drop packets to launch attacks.

Routing misbehavior will significantly reduce the packet delivery ratio and waste the resources of the mobile nodes that have carried and forwarded the dropped packets. To demonstrate this, we simulate the effects of routing misbehavior in two popular DTN routing algorithms, SimBet [4] and Delegation [5] based on the Reality trace [6]. SimBet is a *forwarding-based* algorithm where a packet only has one replica. Delegation is a *replication-based* algorithm where a packet may have multiple replicas. As shown in Fig. 1, when 30% of the nodes with higher connectivity are misbehaving, SimBet only delivers 3% of the packets whereas Delegation only delivers 40%. Moreover, 95% of the transmissions in SimBet and 80% in Delegation are wasted since the packets are finally dropped by misbehaving nodes. Although Burgess *et al.* [7] studied the effects of packet dropping on packet delivery ratio, they did not consider the wasted transmission (bandwidth) caused by dropping. Therefore, it is extremely important to detect packet dropping and mitigate routing misbehavior in DTNs.

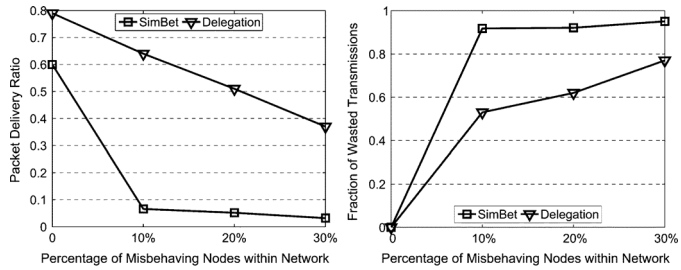


Fig. 1. Effects of routing misbehavior when SimBet and Delegation are used as the routing algorithms.

Routing misbehavior has been widely studied in mobile ad hoc networks [8]–[12]. These works use neighborhood monitoring [8]–[10] or acknowledgement (ACK) [11], [12] to detect packet dropping, and avoid the misbehaving nodes in path selection. However, they do not consider the intermittent connectivity in DTNs and cannot be directly applied to DTNs.

In this paper, we address routing misbehavior in DTNs by answering two questions: how to detect packet dropping and how to limit the traffic flowing to the misbehaving nodes. We first propose a scheme which detects packet dropping in a distributed manner. In this scheme, a node is required to keep previous signed contact records such as the buffered packets and the packets sent or received, and report them to the next contact node which can detect if the node has dropped packets based on the reported records. Misbehaving nodes may falsify some records to avoid being detected, but this will violate some consistency rules. To detect such inconsistency, a small part of each contact record is disseminated to some selected nodes which can collect appropriate contact records and detect the misbehaving nodes with certain probability. Then we propose a scheme to

Manuscript received April 14, 2011; revised August 19, 2011; accepted October 07, 2011. Date of publication October 21, 2011; date of current version March 08, 2012. This work was supported in part by the Army Research Office under MURI Grant W911NF-07-1-0318. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Yong Guan.

The authors are with the Pennsylvania State University, University Park, PA 16802 USA (e-mail: qxl118@cse.psu.edu; gcao@cse.psu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2011.2173195

mitigate routing misbehavior by limiting the number of packets forwarded to the misbehaving nodes.

This paper is structured as follows. Section II reviews related work. Section III introduces our network and security models. Sections IV–VI present the packet dropping detection scheme and analyze its security as well as cost. Section VII presents the routing misbehavior mitigation scheme. Section VIII evaluates of the performance of our solutions. Section IX concludes the paper.

II. Related Work

In mobile ad hoc networks, much work has been done to detect packet dropping and mitigate routing misbehavior. To detect packet dropping, Marti *et al.* [8] proposed watchdog-based solutions in which the sending node operates in promiscuous mode and overhears the medium to check if the packet is really sent out by its neighbor. Some follow-up works [9], [10] have used this neighborhood monitoring approach to detect packet dropping. However, neighborhood monitoring relies on a connected link between the sender and its neighbor, which most likely will not exist in DTNs. In DTNs, a node may move away right after forwarding the packet to its neighbor, and thus cannot overhear if the neighbor forwards the packet.

Another line of work uses the acknowledgement (ACK) packet [11]–[13] sent from the downstream node along the routing path to confirm if the packet has been forwarded by the next hop. Liu *et al.* [11] proposed a 2ACK scheme in which the sending node waits for an ACK from the next hop of its neighbor to confirm that the neighbor has forwarded the data packet. However, this technique is vulnerable to collusions, i.e., the neighbor can forward the packet to a colluder which drops the packet. Although end-to-end ACK schemes [13] are resistant to such colluding attacks, the ACK packets may be lost due to the opportunistic data delivery in DTNs. Moreover, in routing protocols [2], [5], [14] where each packet has multiple replicas, it is difficult for the source to verify which replica is acknowledged since there is no persistent routing path between the source and destination in DTNs.

To mitigate routing misbehavior, existing works [8], [11], [13] in mobile ad hoc networks reduce the traffic flowing to the misbehaving nodes by avoiding them in path selection. However, they cannot be directly applied to DTNs due to the lack of persistent path.

In DTNs, one serious routing misbehavior is the black hole attack, in which a black hole node advertises itself as a perfect relay for all destinations, but drops the packets received from others. Li *et al.* [15] proposed an approach that prevents the forgery of routing metrics. However, if the black hole node indeed has a good routing metric for many destinations, their approach will not work, but our approach still works by limiting the number of packets forwarded to the black hole node. Another related attack is the wormhole attack, which has been recently addressed by Ren *et al.* [16].

To address selfish behaviors, Shevade *et al.* proposed a gaming-based approach [17] and Chen *et al.* [18] proposed a credit-based approach which provide incentives for selfish nodes to forward packets. Li *et al.* [19], [20] proposed a social selfishness aware routing algorithm to allow user selfishness and provide better routing performance in an efficient way. Our work is complementary since besides dealing with selfish routing we also consider the misbehavior of malicious nodes whose goal is not to maximize their own benefits but to launch attacks.

Our solution has some similarity with previous work (e.g., [21]) on detecting node clone attacks in sensor networks, since both detect the attacker by identifying some inconsistency. However, our work relies on a different kind of inconsistency in DTNs, and DTNs do not have the reliable link connection used in existing solutions for node clone attacks.

III. Preliminaries

A. Network and Routing Model

Similar to many other works (e.g., [2], [22]), we assume each node has two separate buffers. One has unlimited space and is used to store its own packets; the other one has limited space and is used to store packets received from other nodes.

We assume the network is loosely synchronized; i.e., any two nodes should be in the same time slot at any time. Since the intercontact time is usually at the scale of minutes or hours, the time slot can be at the scale of one minute. Thus, such loose time synchronization is not hard to achieve.

B. Security Model

There are two types of nodes: *misbehaving nodes* and *normal nodes*. A misbehaving node drops the received packets even if it has available buffers,[1] but it does not drop its own packets. It may also drop the control messages of our detection scheme. We assume a small number of misbehaving nodes may collude to avoid being detected, and they may synchronize their actions via out-band communication channels. A normal node may drop packets when its buffer overflows, but it follows our protocol.

In some DTN applications, each packet has a certain lifetime, and then expired packets should be dropped whether or not there is buffer space. Such dropping can be identified if the expiration time of the packet is signed by the source. Such dropping is not a misbehavior, and will not be considered in the following presentations.

We assume a public-key authentication service is available. For example, hierarchical identity-based cryptography [23] has been shown to be practical in DTNs [24]. In identity-based authentication, only the offline trusted private key generator can generate a public/private key pair, so a misbehaving node itself cannot forge node identifiers (e.g., to launch Sybil attacks). Generally speaking, a node's private key is only known by itself; however, colluding nodes may know each other's private key.

C. Overview of our Approach

mitigate 缓解 减轻

Our approach consists of a packet dropping detection scheme and a routing misbehavior mitigation scheme. Fig. 2(a) illustrates our basic approach for misbehavior detection. The mis-

[1]In the rest of this paper, when we mention "buffer" we mean the buffer that is used to store the packets received from other nodes.

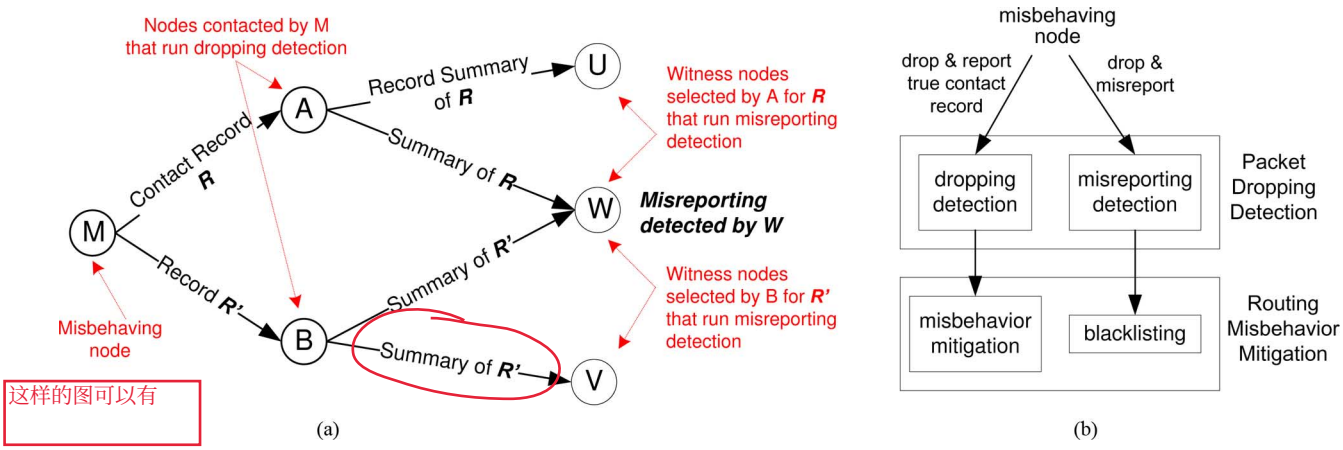


Fig. 2. Overview of our approach. (a) Packet Dropping Detection Misbehaving node M reports two forged contact records R and R' which are inconsistent. (b) Misbehavior Mitigation.

怎么检查到 丢弃报文 ？

必须报告丢弃事件 不报告 就藏不住；
但可以晚一些报告

behaving node [M in Fig. 2(a)] is required to generate a *contact record* during each contact and report its previous contact records to the contacted node [A and B in Fig. 2(a)]. Based on the reported contact records, the contacted node detects if the misbehaving node has dropped packets. The misbehaving node may misreport (i.e., report forged contact records) to hide its misbehavior, but forged records cause inconsistencies which make misreporting detectable. To detect misreporting, the contacted node also randomly selects a certain number of *witness nodes* for the reported records and sends a summary of each reported record to them when it contacts them. The witness node [W in Fig. 2(a)] that collects two inconsistent contact records can detect the misreporting node.

Fig. 2(b) illustrates our approach for routing misbehavior mitigation. It reduces the data traffic that flows into misbehaving nodes in two ways: 1) If a misbehaving node misreports, it will be blacklisted (after the misreporting is detected) and will not receive any packet from other nodes; 2) if it reports its contact records honestly, its dropping behavior can be monitored by its contacted nodes, and it will receive much less packets from them.

我们直接假定它
在TTL来临之前才报告丢弃

D. Terms and Notations

We use N, N_i, N_j, etc. to denote a node, and use M or M' to denote a misbehaving node. We use $\mathrm{SIG}\{*\}$ to denote a node's signature over the content in the bracket. $H(*)$ denotes a cryptographically strong hash function (e.g., SHA-1), and "|" denotes the concatenation operation.

IV. Packet Dropping Detection: A Basic Scheme

This section presents our basic packet dropping detection scheme. We first introduce the basic idea and then describe how to detect packet dropping and how to detect misreporting. Collusions will not be considered in this section, and will be addressed in Section V.

A. Basic Idea

When two nodes contact, they generate a *contact record* which shows when this contact happens, which packets are in their buffers before data exchange, and what packets they send or receive during the data exchange. The record also includes the unique sequence number that each of them assigns for this contact. The record is signed by both nodes for integrity protection.

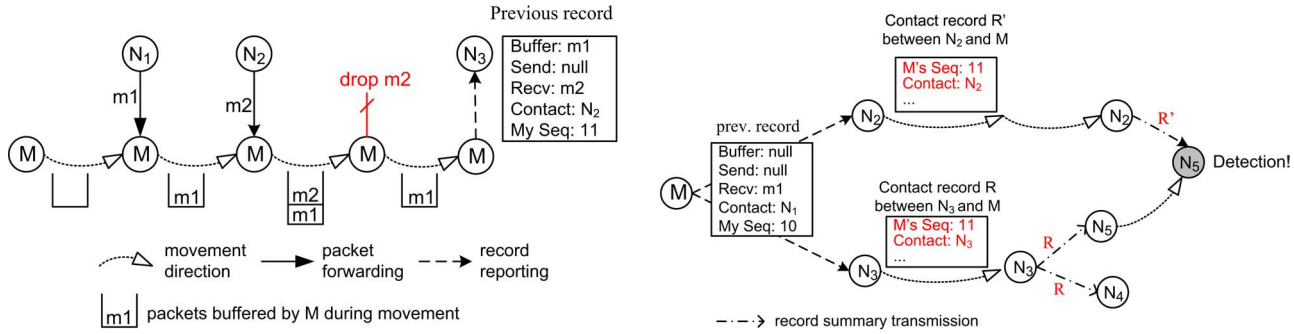
A node is required to carry the record of its previous contact, and report the record to its next contacted node, which will detect if it has dropped packets since the previous contact. We illustrate this with an example shown in Fig. 3(a). M buffers packet $m1$ before the contact with node N_2 and it receives $m2$ from N_2 in the contact. Such information is included in the contact record that M reports to its next contacted node N_3. From the record N_3 can deduce that M should still buffer $m1$ and $m2$ at the current time. If M has dropped $m2$ after the contact with N_2, N_3 will detect the dropping when it exchanges the buffer state with M.

怎么检查出已经丢弃了呢？ 完整性？//检查了公私钥？

A misbehaving node may report a false record to hide the dropping from being detected. However, misreporting will result in inconsistent contact records generated by the misbehaving node. To detect misreporting, for each contact record that a normal node generates with (or receives from) other nodes, the normal node selects w witness nodes and transmits the record summary to them. The summary only includes a part of the record necessary for detecting the inconsistency caused by misreporting. With some probability, the summaries of two inconsistent contact records will reach a common witness node which will detect the misreporting node.

一部分record summary

We use the example in Fig. 3(b) to show how misreporting causes inconsistency and how it can be detected. After dropping $m2$, M does not report the contact record with N_2 to N_3, but it dishonestly reports the contact record with N_1 as the "previous" record to N_3 as if it did not contact N_2. Based on this record N_3 cannot detect the dropping of $m2$. Since this record is reported to both N_2 and N_3, M has to assign the same sequence number to the contact with N_2 and N_3, which is one plus the sequence number it assigned to the reported "previous" record. Since the same sequence number should not be assigned to two different contacts, the two records that M generates with N_2 and N_3 are inconsistent. Suppose both N_2 and N_3 have selected N_5 as the witness node for their contact record. When N_5 receives the summaries of the two inconsistent records from $N2$ and $N3$, it will detect the misreporting of M.



(a) (b)

Fig. 3. Examples of packet dropping detection and misreporting detection. In (b), M reports the same contact record to N_2 and N_3 as its "previous" record, and then it has to assign the same sequence number to the contact with N_2 and N_3, which violates consistency rules.

前一个contact之前就有 或者 前一个contact中接收了；但这次contact之前没有 也没在前一个contact里发送出去

B. Contact Record and Record Summary

Suppose a contact happens between node N_i and N_j at time t. Without loss of generality, suppose $i < j$. Let BV_i and BV_j denote the vector of packets buffered by N_i and N_j before this contact, respectively. Let RV_i and RV_j denote the identifiers of the packets received by N_i and N_j during this contact, respectively. Then the record of this contact that N_i will store and report is as follows:

$$\begin{aligned}\mathcal{R} &= N_i, sn_i, N_j, sn_j, t, BV_i, RV_i, RV_j, \mathrm{sig}_i^1, \mathrm{sig}_j^1, \mathrm{sig}_i^2, \mathrm{sig}_j^2\\ \mathrm{sig}_{i,j}^1 &= \mathrm{SIG}_{i,j}\left\{H\left(N_i|sn_i|N_j|sn_j|t|BV_i\right)\right\}\\ \mathrm{sig}_{i,j}^2 &= \mathrm{SIG}_{i,j}\left\{H\left(N_i|sn_i|N_j|sn_j|t|H(RV_i)|H(RV_j)\right)\right\}.\end{aligned} \tag{1}$$

sn_i and sn_j are the sequence numbers assigned by N_i and N_j for this contact, respectively. A node assigns sequence number 1 to its first contact, and increases the number by one after each contact. The sequence number is large enough (e.g., 32 bits) so that a node will not use the same number twice. The summary of this record is

$$\mathcal{S} = N_i, sn_i, N_j, sn_j, t, H(RV_i), H(RV_j), \mathrm{sig}_i^2, \mathrm{sig}_j^2. \tag{2}$$

The size of record summary is constant, no matter how many packets are buffered before or exchanged during the contact. Note that the record of this contact that N_j will store and report is similar to that by N_i except that BV_i is replaced with BV_j.

If the two nodes contact each other several times without contacting any other node, the transactions in the second and later contacts can be seen as an extension of the transaction of the first contact. Then, only one record is generated in the first contact, and its RV and the appropriate signature components are updated in later contacts.

前一个contact之前 S_bgn
现在这个contact之前 S_end

C. Packet Dropping Detection

In a contact, each of the two contacting nodes reports its previous contact record [see (1)] to the other node. In this contact, the two nodes also exchange their current vector of buffered packets (as a step of contact record generation). In this way, one node knows the two sets of packets the other node buffers at the beginning of the previous contact and the beginning of the current contact, which are denoted by $\mathbb{S}_{\mathrm{bgn}}$ and $\mathbb{S}_{\mathrm{end}}$, respectively. It also knows the two sets of packets the other node sends and receives in the previous contact, which are denoted by $\mathbb{S}_{\mathrm{snd}}$ and $\mathbb{S}_{\mathrm{rcv}}$, respectively. If forwarding-based routing is used, the other node has dropped packets iff

$$\exists\, m \in \mathbb{S}_{\mathrm{bgn}} \cup \mathbb{S}_{\mathrm{rcv}},\ m \notin \mathbb{S}_{\mathrm{end}} \quad \text{and} \quad m \notin \mathbb{S}_{\mathrm{snd}}. \tag{3}$$

If replication-based routing is used, the other node has dropped packets iff

前一个contact之前就有 或者 前一个contact中接收了；但这次contact之前没有

$$\exists\, m \in \mathbb{S}_{\mathrm{bgn}} \cup \mathbb{S}_{\mathrm{rcv}},\ m \notin \mathbb{S}_{\mathrm{end}}. \tag{4}$$

A misbehaving node may drop a packet but keep the packet ID, pretending that it still buffers the packet. However, the next contacted node may be a better relay for the dropped packet according to the routing protocol, which can be determined when the two exchange the destination (included in packet ID) of the buffered packets. In this case, the misbehaving node should forward the packet to the next contacted node, but it cannot since it has dropped the packet. Thus, the next contacted node can easily detect this misbehavior and will not forward packets to this misbehaving node.

关于报文的真实性 //另一个问题

D. Misreporting Detection

Suppose a misbehaving node M has dropped some packets. To hide the dropping from being detected by the next contacted node, M will not report the true record of the previous contact. However, when there is no collusion, M cannot modify the true record since it is signed by the previous contacted node. Also, M cannot forge a contact record because it does not know the private key of any other node. Thus, the only misreporting it can perform is to replay an old record generated before the previous contact as illustrated in Fig. 3(b). This misreporting is referred to as *replay-record*. Note that other types of misreporting are possible when collusions exist, and we will address them in Section V.

把record重放

1) Consistency Rules: There exist two simple rules which are obeyed by normal nodes but violated by misreporting nodes:

- Rule 1: Use a unique sequence number in each contact.
- Rule 2: For two records signed by the same node, the record with a smaller contact time also has a smaller sequence number.

With replay-record, the misreporting node violates Rule 1 and/or Rule 2. In the example shown in Fig. 3(b), M replays the record that it has reported to N_2 when it contacts N_3, and it has to assign the same sequence number (i.e., 11) for the contacts with N_2 and N_3, which means Rule 1 is violated. If M replays

我方节点 可以知道 //对方 previous contact之前 和 本次contact之前 报文// previous contact中 发送 和 接收的报文

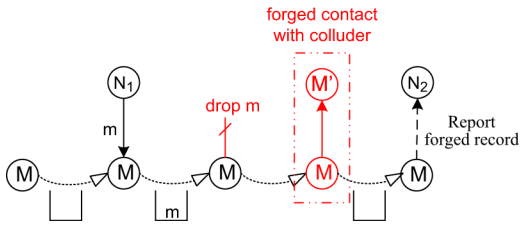


Fig. 4. Two colluding nodes M and M' try to hide the dropping of packet m by forging a contact (via the out-band channel) and reporting the forged contact record. The forged record may show that M has not received m, M has forwarded m to M', etc.

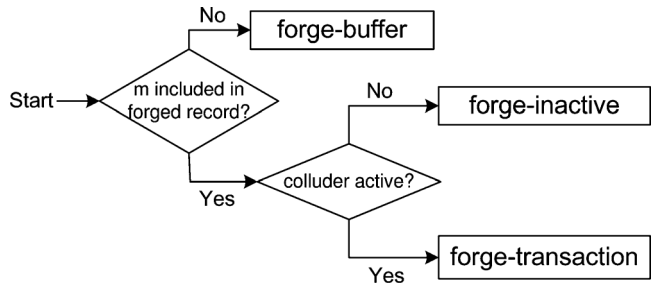


Fig. 5. Decision-tree based exploration of misreporting.

an earlier record, the sequence number it assigns for the contact with N_3 will be smaller than the number it assigns for the contact with N_2, which means Rule 2 is violated. For a special case where the misbehaving node does not report any previous record pretending that it has not had any contact, every time it does so, it has to claim that the current contact is its first contact and assign 1 as its sequence number. Then, Rule 1 is violated.

2) Detection: To detect the inconsistency caused by misreporting, for each contact record generated and received in a contact, a node selects w random nodes as the witness nodes of this record, and transmits the summary of this record to them when it contacts them. It selects the witness nodes from the nodes that it has directly contacted. Here, the nodes contacted a long time ago are not used since they may have left the network.

In this manner, each node can collect some record summaries for which it is a witness. When it receives a new summary, it checks the summary against the already collected summaries signed by the same node to see if the signer has violated any of the two consistency rules. If a violation is detected, it further verifies the signatures included in the inconsistent summaries. If the signature verification succeeds, the signer is detected as misreporting.

3) Alarm: After detection, the witness node floods an alarm (via Epidemic routing [25]) to all other nodes. The alarm includes the two inconsistent summaries. When a node receives this alarm, it verifies the inconsistency between the included summaries and the signature of the summaries. If the verification succeeds, this node adds the appropriate misreporting node into a blacklist and will not send any packets to it. If the verification fails, the alarm is discarded and will not be further propagated. A misreporting node will be kept in the blacklist for a certain time before being deleted.

关于 misreporting: witness验过之后 alarm到各个节点；每个节点本地 还会再次验

V. Dealing With Collusions

This section extends the basic scheme presented in Section IV to address the collusions among misbehaving nodes.

A. Misreporting With Collusion

Suppose a misbehaving node M drops a packet. If M reports the true record of its previous contact to the next contacted (normal) node, the dropping can be detected. To hide the dropping, M can forge a contact with its colluder M' after dropping the packet, and report the falsified contact record to the next contacted node, as illustrated in Fig. 4.

The two colluding nodes can hide the dropping in several ways. We use a decision tree to describe the possible collusions, as shown in Fig. 5. The following explanations are based on the example shown in Fig. 4.

M and M' first need to decide if the forged record lists the dropped packet m as being buffered by M. If not, M will tell N_2 that it did not receive m, and N_2 cannot detect the dropping of m. This misreporting behavior is referred to as *forge-buffer*.

If the forged record lists m as being buffered by M, then to hide the dropping, the forged record should also show that M has forwarded m to M' in a forged contact. M is safe to report this record to N_2 without being detected, but if M' reports this record to its next contacted (normal) node, M' will be detected as having dropped m. Thus, M' cannot report such a record.

There are two choices for M' to avoid being detected. 1) M' does not report any record to noncolluding nodes in the future. Then, it has to keep inactive, e.g., turning off its wireless transmitter. This misreporting behavior is referred to as *forge-inactive*. 2) M' keeps active and it reports a record of the forged contact to its next contacted node, but the record reported by M' is different from the one reported by M. This means that M and M' generate two different records for the same forged contact. This misreporting behavior is referred to as *forge-transaction*.

In forge-transaction, two different records are generated for the same forged contact. Thus, forge-transaction violates Rule 1 and can be detected by the basic scheme presented in Section IV. In the following, we address other collusions.

两种不同的forge

B. Forge-Buffer

Forge-buffer can hide packet dropping because the normal nodes contacted by the misbehaving node and its colluder cannot see a long-enough contact history of them. To address this problem, the basic scheme is extended such that each node is required to report more than one previous record in its contact. The number of records the node needs to report is determined by its current *report window*.

Let $r(r \geq 2)$ denote the number of colluding nodes. The report window of a node includes the minimum number of its most recent contacts so that it includes at least r different nodes. As shown in Fig. 6 $(r = 2)$, when contacting N_2, M's report window includes the contact with N_0 and the contact with N_1 since these two contacts are with two different nodes. However, when contacting N_4, M's report window includes the contact with N_2 and the two contacts with N_3. Due to the opportunistic nature of contacts, the average size of the report window is

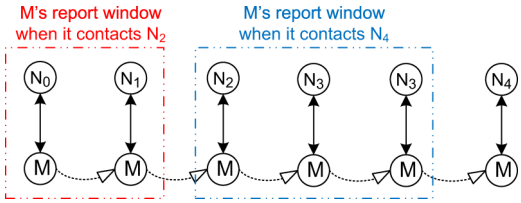


Fig. 6. Examples of report window when $r = 2$. A node is required to report the records of the contacts in its current report window to the next contacted node.

usually close to r. For the Reality DTN trace [6], the average window size is 2.34 when $r = 2$.

During a contact, the two contacting nodes exchange records of the contacts in their report window. In this way, a node knows more history information about the packets that the contacting node buffers, sends and receives. Then it can detect if the other node has dropped packets during the report window similar to that in the basic scheme (see Section IV-C). Since the contacts in a report window happen successively, a node does not need to report the vector of its buffered packets in the intermediate contacts of the window, which can reduce the communication cost.

In forge-buffer, since the misreporting node is required to report the recent records with at least r distinct nodes but it only has $r - 1$ colluders, the record of its previous real contact has to be reported. Then it is trivial for the contacted node to detect the packet dropping, and forge-buffer fails.

C. Forge-Inactive

To address forge-inactive, a forwarding rule is added which requires that packets can only be forwarded to an *active node*, which is defined as the node that contacts at least r different nodes within the recent time interval T_{active}.[2]

To support the forwarding rule, each node maintains an *activeness table* which includes the active nodes it knows from local knowledge. Each table entry has three elements, the identifier of an active node, the most recent time this active node has proved to be active, and the activeness proof. The activeness proof contains part of the active node's most recent r contact records with r different nodes. An entry expires when T_{active} has passed since the time given by the second element. Then the entry is deleted and the active node is tagged as inactive.

The activeness of a node can be easily proved by the recent contact records it reports to other nodes. During a contact one node is supposed to be able to obtain the contacted node's most recent contact records with r nodes. Based on these records, it updates its activeness table entry for the contacted node. If it has some packets to the contacted node, it forwards them only when the contacted node is active. Even if the contacted node is inactive, the two nodes still generate a record for this contact although the packet is not forwarded. This will help newly joined nodes quickly accumulate contact records and quickly become a packet forwarder.

The two nodes also check if anyone has broken the rule and forwarded packets to inactive nodes. Suppose from the reported contact records N_i knows that N_j has forwarded packets to node N_k. If N_k is not active from the local knowledge of N_i, then N_j should show N_i the activeness proof of N_k. If N_j cannot provide a valid proof, N_i knows that N_j has invalidly forwarded packets to an inactive node, and it adds N_j into the blacklist.

In forge-inactive, M' becomes inactive after the forged contact. Although M can still "forward" packets to M' within the next T_{active}, it cannot continue to do so after T_{active} due to the forwarding rule. Then forge-inactive becomes useless to M.

The forwarding rule has a side effect: packets will not be forwarded to the nodes which are not able to contact more than $r-1$ other nodes. Since these nodes' forwarding capability cannot be exploited, packet delivery ratio may be reduced. However, when r is small, these nodes have limited forwarding capability and probably there are not too many of them. Thus, the side effect on packet delivery ratio should be marginal.

D. Record and Summary Deletion

A node deletes the record that it generates in a contact after the contact has been purged out of its report window, probably after a few contacts. It deletes the records received from the contacted node right after this contact, since these received records are only used to check if the contacted node has dropped packets recently.

The witness node should keep its collected record summaries for a long enough time to detect misreporting. For simplicity, our scheme uses a time-to-live parameter T_{delete}, which denotes the time for the collected summaries to be stored before being deleted.

In forge-inactive, the colluder M' may become active again after being inactive for some time, and then perform forge-transaction. If the inactive time is longer than T_{delete}, this misreporting may not be detected, since the record reported by M which is inconsistent with the record reported by M' may have been deleted by the witness nodes. Considering this problem, a witness node can keep the most recent r record summaries generated by each inactive node even if these summaries have expired. The witness nodes that receive the summary of the record reported by M will keep the summary and (with certain probability) detect the misreporting when M' becomes active again.

E. Discussion

M-M'-M''

After M tells the next contacted node that it has forwarded the dropped packet m to the colluder M', M' may tell its own next contacted node that it has forwarded m to M or another colluder M''. This is equivalent to the case that M' has dropped m and it wants to hide the dropping with the help of M or M''. Such fake forwarding can continue among the colluding nodes.

When only two nodes collude ($r = 2$), this phenomenon can be easily detected with report window. Since the forged contacts between the two colluding nodes are within their report window, their next contacted node can detect that m is forwarded back-and-forth between the two colluding nodes.

However, the report window may not work when three or more nodes collude because a report window may not include all the colluding nodes in the chain of fake forwarding. A more generic solution is needed. Since the number of colluders is limited and small, the chain of fake forwarding cannot extend forever and it will loop back. In the above example, suppose there

forge-inactive 长时间不报告，没有对应的未超时 entry 不会给它传报文

[2]The packets destined to a node which can only contact $r - 1$ other nodes can still be delivered to it.

are only three colluders. Then M'' will have to claim that it has forwarded m back to M or M'. This means that at least one colluding node in the chain will "receive" m twice or more. However, in most routing protocols a normal node will not receive the same packet more than once. Therefore, repeated receiving of the same packet can be used to detect such fake forwarding. From the reported contact records, nodes can monitor each other on the packets they have received, and raise an alarm when repeated receiving of the same packet is detected.

VI. Analysis of Misreporting Detection

In our scheme, a misreporting node can be detected when a witness node receives two inconsistent summaries generated by this misreporting node. Due to the opportunistic nature of DTN, the detection is also opportunistic. In this section, we analyze the probability of detection and the detection delay. We also analyze the cost of the detection scheme.

A. Models and Notations

Poisson Process的假设

For simplicity, we base our analysis on the mobility models such as Random Waypoint and Random Direction where the contacts between node pairs can be modeled as i.i.d. Poisson processes [26]. Then a node will encounter any other node in its next contact with the same probability. Poisson contact model has been experimentally validated by [27]–[29] to fit well to realistic DTN traces. In the analysis, we assume a node can disseminate a record summary to its selected witness nodes, and the witness nodes store the summary for a long enough time.

Let n denote the number of nodes in the network, and q denote the proportion of misbehaving nodes. Without loss of generality, the following analysis considers forge-transaction in which misreporting node M and its colluder M' generate two inconsistent contact records $\mathcal{R}$ and $\mathcal{R}'$. M reports $\mathcal{R}$ to its next two contacted nodes which constitute a set $\mathbb{S}$, and M' reports $\mathcal{R}'$ to its next two contacted nodes which constitute a set $\mathbb{S}'$.

For convenience, we define function $P_o(x, y)$ as the probability that two randomly and independently selected sets of nodes with size x and $y (x, y < n)$ have at least one common node. It is trivial to get $P_o(x, y) = 1 - \binom{n-x}{y} / \binom{n}{y}$.

B. Detection Probability

正常在n里选中y个； 在n-x里选中y个

We first analyze the probability P_d that a single misreporting instance can be detected. If $\mathbb{S}$ overlaps with $\mathbb{S}'$ and the overlapping node is a normal node, the attack can be detected by the overlapping node. This probability is given by $P_o(2, 2) = 4n - 6/n(n-1)$. Since n is usually large, it is negligible. In the following, we consider the case where $\mathbb{S}$ and $\mathbb{S}'$ do not overlap.

The detection succeeds when: 1) there is at least one normal node in $\mathbb{S}$ and $\mathbb{S}'$; 2) the normal nodes in $\mathbb{S}$ and those in $\mathbb{S}'$ select one or more common witness nodes for the two records; and 3) at least one common witness is a normal node. Obviously, the three conditions hold independently. The probability that the first two conditions hold depends on how many nodes in $\mathbb{S}$ and $\mathbb{S}'$ are normal. In the case that both the two nodes in $\mathbb{S}(\mathbb{S}')$ are normal, the probability that they select a common witness for $\mathcal{R}(\mathcal{R}')$ is given by $P_o(w, w)$, which is small since $w \ll n$. For simplicity, in this case, we assume their selected witness nodes do not overlap, i.e., they select $2w$ witness nodes for $\mathcal{R}(\mathcal{R}')$.

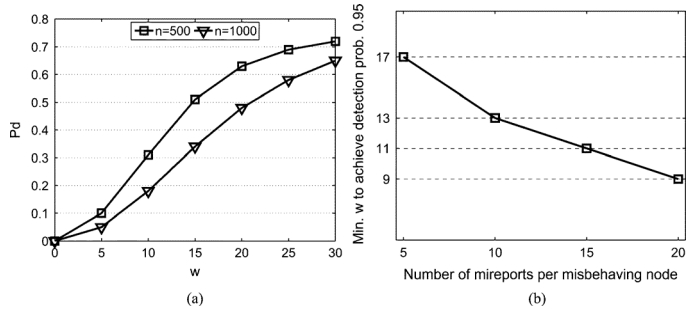


Fig. 7. Numerical results on misreporting detection where 20% of nodes are misbehaving (i.e., $q = 0:2$). (a) Detection probability when each misbehaving node misreports once. (b) The minimum w required to detect each misreporting node with probability not less than 0.95 when $n = 1000$.

Then the probability that the first two conditions hold is given by

4种情况讨论

$$P_o(w, w) \cdot [2q(1-q)]^2 + P_o(w, 2w) \cdot [4q(1-q)(1-q)^2] + P_o(2w, 2w) \cdot (1-q)^4.$$

Since the third condition holds with probability not smaller than $1 - q$, the detection probability has a lower bound

$$P_d = P_o(w, w) \cdot 4q^2(1-q)^3 + P_o(w, 2w) \cdot 4q(1-q)^4 + P_o(2w, 2w) \cdot (1-q)^5.$$

Numerical results on P_d are shown in Fig. 7(a). P_d increases as the number of witness nodes w increases, and it is higher than 0.5 even when w is smaller than the square root of n.

If a high P_d is required, w is not very small. However, in reality, it may not be necessary to detect each misreporting instance (which requires a high P_d). A misbehaving node needs to drop packets many times to make a reasonable impact and misreport many times to hide its misbehavior. A good-enough security can be provided if the misbehaving node can be detected with a high probability after a reasonably small number of misreporting instances. Suppose a misbehaving node launches k misreporting instances independently. Then the node will be detected with probability $P' = 1 - (1 - P_d)^k$, which is much higher than P_d when $k > 1$. Given a certain required probability P' (e.g., 0.95), we can find the minimum w with which the misbehaving node can be detected with probability not less than P' after it misreports for k times. Fig. 7(b) shows the minimum w required to detect misbehaving nodes with probability not less than 0.95 when different numbers of misreporting instance k can be tolerated. Basically, w will be significantly reduced if k increases.

C. Detection Delay

Detection delay (T_d) is defined as the elapsed time from when $\mathcal{R}$ and $\mathcal{R}'$ have been reported to the nodes in set $\mathbb{S}$ and $\mathbb{S}'$, respectively, to the time when the misreporting instance is detected by a witness node. Detection delay is meaningful only when misreporting can be detected.

If a misreporting is detected, there is at least one witness node which receives the summary of both $\mathcal{R}$ and $\mathcal{R}'$. When there is only one such witness node, the detection delay is the time needed for the summary of $\mathcal{R}$ and $\mathcal{R}'$ to be transmitted to *this* witness node; when there are multiple witness nodes, detection

delay is the time needed for the summary of $\mathcal{R}$ and $\mathcal{R}'$ to be transmitted to *any* of these witness nodes. Since the contact patterns between different node pairs are i.i.d., the expected delay is longer in the former case. Let T_d' denote the detection delay in that case. We derive the expectation of T_d' to obtain an upper bound of the expected T_d.

Let λ denote the Poisson contact rate. Then the cumulative distribution function (cdf) of intercontact time between any node pair is $F_c(t) = 1 - \exp(-\lambda t)$. Since the witness node receives the summary of $\mathcal{R}$ and $\mathcal{R}'$ independently, the cdf of T_d' is

$$F_d(t) = (1 - \exp(-\lambda t))^2$$

and the probability density function (pdf) of T_d' is

$$f_d(t) = F_d'(t) = 2\lambda(1 - \exp(-\lambda t))\exp(-\lambda t).$$

Then the expectation of T_d' is

$$E(T_d') = \int_0^{+\infty} 2\lambda t(1 - \exp(-\lambda t))\exp(-\lambda t)dt = \frac{3}{2\lambda}.$$

Now we have an upper bound for the expected detection delay

对 pdf 进行积分 得到平均延迟

$$E(T_d) \leq \frac{3}{2\lambda}.$$

D. False Positive

Our misreporting detection scheme does not have false positive when detecting replay-record and forge-transaction. This is because normal nodes do not violate any of the consistency rules.

An attacker may spoof another normal node and insert new inconsistent contact records or record summaries on behalf the normal node, so that when the witness receives the inconsistent record summaries it detects the normal node as a misreporting attacker. However, in our scheme both contact record and record summary are protected by the signature of the node that generates them. Since the attacker does not know the private key of any normal node (as our assumption), it cannot forge a valid contact record or record summary.

E. Cost

We analyze the computation cost per contact in terms of signature generations and verifications. In each contact, the two nodes invocate six signature generations and verifications to generate the contact record. The reported records require $8r$ signature verifications in total (assuming one report window contains r contacts). To enforce the forwarding rule, each node in the worst case verifies $2r$ signatures (i.e., $4r$ in total). Also, the summary of each contact is disseminated to $4w$ witness nodes. Note that the witness nodes verify the signatures of the summary only when some violation is detected. If there is no misreporting node, there is no signature verification. To sum up, each contact requires 6 signature generations and at most $12r + 6$ signature verifications. This cost is low considering that signature verification is usually much faster than signature generation.

For the communication cost, in each contact, the buffer states of the two nodes are exchanged, a few contact records are reported, and a few activeness proofs may be transmitted. Each of the contacting nodes transmits the two reported summaries for at most w times. Since only meta data is transmitted and for very limited time, the communication overhead is low. The storage cost mainly comes from the record summaries. Since each summary only has around 100 bytes [see the setting in Section VIII-D3 and equation (2)] and will be deleted later, the storage cost is also low.

Algorithm 1 FP Update (run by N_i when it contacts N_j)

Require: The gauge list GL that N_i maintains form N_j
Require: The r nodes $N_x(1 \leq x \leq r)$ contacted by N_j in its report window
Require: $drop$ = FALSE, $receive$ = FALSE, $forward$ = FALSE
Require: $0 < \delta < 1, 0 \leq \rho < 1$

Delete the two oldest elements of GL
Insert $N_x(1 \leq x \leq r)$ into GL
if N_x is the most-frequent element of GL but not N_i itself **then**
    Tag N_x as *special*
else
    Tag N_x as *plain*
end if
if N_j has dropped packets in the report window **then**
    $drop$ = TRUE
end if
if N_j has received packets from a *plain* node (N_x) **then**
    $receive$ = TRUE
end if
if N_j has forwarded packets to a *plain* node (N_x)**then**
    $forward$ = TRUE
end if
if $drop$ == TRUE
    $\gamma = \gamma \cdot \rho$
else if $receive$ == TRUE or $forward$ == TRUE
    $\gamma = \min\{\gamma + \delta, 1\}$
else
    $\gamma = \max\{\gamma - \delta, 0\}$
end if

VII. Routing Misbehavior Mitigation

To mitigate routing misbehavior, we try to reduce the number of packets sent to the misbehaving nodes. If a node is detected to be misreporting, it should be blacklisted and should not receive packets from others. However, if a misbehaving node does not misreport, we cannot simply blacklist it because it is dropping packets, since a normal node may also drop packets due to buffer overflow. In the following, we focus on how to mitigate routing misbehavior without affecting normal nodes too much when misbehaving nodes do not misreport.

Our basic idea is to maintain a metric *forwarding probability (FP)* for each node based on if the node has dropped, received and forwarded packets in recent contacts, which can be derived from its reported contact records. The nodes that frequently drop packets but seldom forward packets will have a small FP and will receive few packets from others. Our scheme borrows ideas

总是丢包的 可以少发；

from congestion control to update FP. More specifically, it combines additive increase, additive decrease, and multiplicative decrease to differentiate misbehaving nodes from normal nodes.

A. FP Maintenance

Each node maintains an FP for every other node it has contacted based on the local knowledge obtained from previous contacts, and updates the FP with new contacts. Let $\gamma(0 \leq \gamma \leq 1)$ denote the FP that node N_i maintains for N_j. When N_i contacts N_j, it obtains the recent contact records of N_j which are used to derive if N_j has dropped, received, and forwarded packets in the report window. If N_j has dropped packets, N_i decreases γ by a multiplicative factor $\rho(0 \leq \rho < 1)$. If N_j has not dropped packets, there will be two subcases. If N_j has received packets or forwarded packets out, N_i knows that N_j has contributed its buffer or bandwidth to the network, and it increases γ by an additive amount $\delta(0 < \delta < 1)$. If N_j has not received or forwarded any packets, it can either be a misbehaving node or a normal node. For security concerns, N_i conservatively takes N_j as misbehaving, and decreases γ by δ.

看Nj和谁关系密切

A misbehaving node may "forward" packets to or "receive" packets from its colluder via the out-band channel, to deceive its contacted nodes to increase the FP for it. To address this problem, node N_i keeps a *gauge list* for every other node N_j it has contacted, which includes the nodes contacted by N_j that N_i observes in the recent c contacts with N_j. If N_j frequently launches the above collusion behavior, its colluder will appear frequently in the gauge list. When N_i observes that N_j has forwarded (received) packets in the report window, N_i checks if the packets are forwarded to (received from) the top $r-1$ nodes which appear most frequently in the gauge list. If this is true, it is very likely that N_j has "forwarded" packets to ("received" packets from) its colluder, and N_i will not increase the FP for N_j. Here, r is the number of colluders. When r is small, we use $c = 10$.

出现得最r-1个频繁的

The initial value of FP is recommended to be set smaller than 1, which means a node from the start does not fully trust other nodes but it gradually builds the trust on them. We set the initial FP as 0.6. Experimentally, we found out that a lower ρ which means a fast decrement, coupled with a low δ which means a slow increment can result in good performances. We use $\rho = 0.1$ and $\delta = 0.1$. The pseudocode of FP update is shown in Algorithm 1.

B. Dealing With Packet Dropping

Suppose node N_i has selected a set of packets (according to the routing algorithm) to forward to its contacted node N_j. Let $\mathbb{S}_{\text{opt}}$ denote this set of packets. Then N_i forwards them with the following policy:

$$\text{For } m \in \mathbb{S}_{\text{opt}}, \text{ forward } m \text{ with probability } \gamma.$$

Our scheme can effectively mitigate routing misbehavior, since if a node is misbehaving and frequently drops packets, it will be assigned a low FP by its contacted nodes and receive few packets from them. Our scheme has the minimal (or no) effect on the normal nodes that seldom or never drop packets. For the "hot spot" normal nodes that receive many packets from others and frequently drop them due to buffer overflow, our scheme will reduce the amount of packets forwarded to them, and thus relieve the congestion at these nodes.

VIII. Performance Evaluations

A. Experiment Setup

We evaluate our solutions both on a synthetic trace generated by the Random Waypoint (RWP) model [26] and on the MIT Reality trace [6] collected from the real world.

In the synthetic trace, 97 nodes move in a 500×500 area with the RWP model. The moving speed is randomly selected from $[1, 1.6]$ to simulate the speed of walking, and the transmission range is 10 m to simulate that of Bluetooth. Each simulation lasts 5×10^5 time units.

50w个time; 11w个contact

In the Reality trace, 97 phones are carried by students and staff at MIT over 10 months. These phones run Bluetooth device discovery every 5 min and log about 110 thousand contacts with each other. Each logged contact includes the two contact parties, the start-time, and the contact duration. We use these contacts to generate trace-driven simulations.

In our simulations, misbehaving nodes are either *randomly deployed* or *selectively deployed* . In the latter case, they are the high-connectivity nodes which have the most frequent contacts with others. Misbehaving nodes drop all or part of the packets they receive from others. In default, each normal node generates one packet (with size 10 KB) every day to a random destination. A misbehaving node does not generate packets as done in [7] and [15]. Each node has buffer size of 5 MB and bandwidth of 2 Mb/s. Results are averaged over 10 runs with different random seeds.

B. Routing Algorithms

SimBet [4]: a forwarding-based routing algorithm. It calculates a metric using two social measures (similarity and betweenness), and a packet is forwarded to a node if that node has higher metric than the current one.

Delegation [5]: a replication-based routing algorithm, where the receiving node replicates the packet to a neighbor if the neighbor has the highest utility it has seen. We use the contact frequency with the destination as the utility metric.

C. Metrics

- **Packet delivery ratio**: The percentage of packets delivered to their destinations out of all generated packets.
- **Number of wasted transmissions**: In forwarding-based routing, a transmission is wasted if the transmitted packet is dropped by misbehaving nodes before reaching the destination; in replication-based routing, a transmission is wasted if it directly transmits a packet to a misbehaving node that drops the packet. Here, we only count the wasted packets dropped by routing misbehavior.
- **Detection rate**: The percentage of misreporting nodes detected by normal nodes.
- **Detection delay**: The time needed for misreporting to be detected.
- **Bytes transmitted per contact**: The number of bytes transmitted in a contact for control.

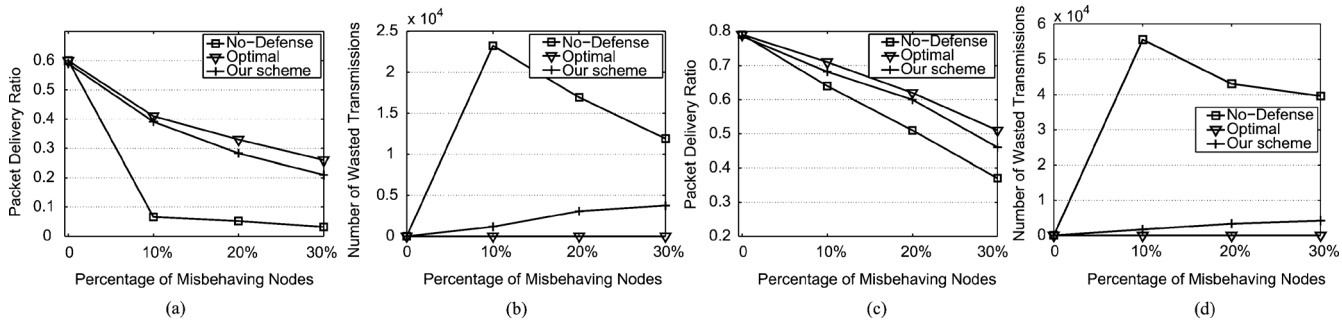


Fig. 8. Comparison results when misbehaving nodes are selectively deployed to high-connectivity nodes which drop all received packets. The Reality trace is used. (a) SimBet. (b) SimBet. (c) Delegation. (d) Delegation.

变成一个3维图吗? 还是 画一个平面
anyway, 先计算一种情况

- **Bytes stored per node**: The number of bytes stored at a node for control.

D. Experimental Results

We compare our scheme to two solutions: one is called *No-Defense*, which does not deal with routing misbehavior; the other one is the *optimal* scheme, which assumes that all misbehaving nodes are known and no packet will be forwarded to them.

1) Routing Misbehavior Mitigation: We first evaluate the case where misbehaving nodes drop all received packets. Fig. 8 shows the comparison results on the Reality trace. Generally speaking, our scheme performs much better than No-Defense. For SimBet, where the routing performance is the major concern, our scheme is close to the optimal in terms of packet delivery ratio; for Delegation, where the reduction of wasted transmission is the major concern, our scheme is close to the optimal in terms of wasted transmissions.

可以这样说

When SimBet is used as the routing algorithm, as shown in Fig. 8(a), the packet delivery ratio of all three schemes decreases as the percentage of misbehaving nodes increases, because fewer nodes can be used for packet forwarding. However, our scheme still delivers much more packets than No-Defense, since it can effectively limit the number of packets forwarded to misbehaving nodes. For similar reasons, our scheme has a much lower number of wasted transmissions than No-Defense, as shown in Fig. 8(b). In No-Defense, the number of wasted transmissions first increases and then decreases as the percentage of misbehaving nodes increases. This is because with more misbehaving nodes, more packets are dropped but the average number of hops traversed by each dropped packet is smaller. As a result, the maximum is reached at some middle point.

When Delegation is used as the routing algorithm, again, our scheme delivers more packets than No-Defense. As shown in Fig. 8(c), when 30% of nodes are misbehaving, our scheme delivers 46% of packets which is 24% higher than that of No-Defense. No-Defense performs worse due to the following reason. In Delegation, although replicating a packet to a misbehaving node does not decrease the probability of other replicas carried by normal nodes to reach the destination, it can reduce the probability of the packet to be further replicated. As a result, fewer replicas are created and the overall probability of reaching the destination is reduced. For the number of wasted transmissions as shown Fig. 8(d), our scheme performs much better than No-Defense.

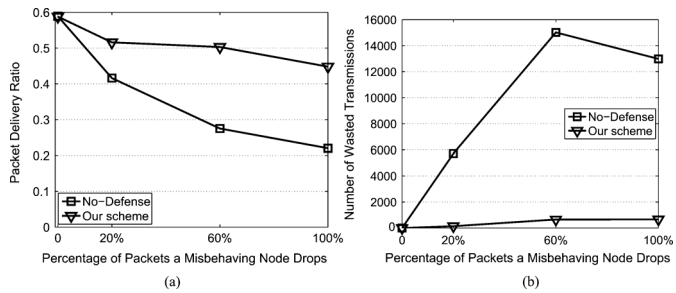


Fig. 9. Comparison results when misbehaving nodes are randomly deployed and they only drop part of the received packets. The Reality trace is used, and the fraction of misbehaving nodes is fixed at 30%.

Then we evaluate the case where misbehaving nodes only drop part of the received packets. Fig. 9 shows the comparison results when SimBet is used as the routing algorithm on the Reality trace. When misbehaving nodes only drop part of the received packets, *optimal* does not necessarily perform the best since some packets can be delivered by misbehaving nodes. Thus, *optimal* is not compared here. As shown in Fig. 9(a), the packet delivery ratio of No-Defense decreases from 59% to 22% as the percentage of received packets that a misbehaving node drops increases from 0% to 100%, because more packets are dropped. The packet delivery ratio of our scheme is much higher than that of No-Defense. For example, when misbehaving nodes drop 60% of the received packets, our scheme delivers 50% of the generated packets, and it outperforms No-Defense by 80%. This shows that our scheme can still effectively limit the number of packets forwarded to misbehaving nodes. For the number of wasted transmissions as shown in Fig. 9(b), our scheme performs much better than No-Defense.

2) Misreporting Detection: In this group of simulations, 10 pairs of misbehaving nodes (i.e., 20 in total) launch forge-transaction independently.

First of all, we verify our analysis results on the detection probability and detection delay of a single misreporting instance (i.e., P_d and T_d in Section VI). The synthetic trace is used since it is accordant to the mobility assumption in Section VI. In each run, 200 misreporting events are generated and detected independently. As shown in Fig. 10, the detection probability in the simulations is higher than the analytical lower bound, but the difference is small which means the analytical result is a good

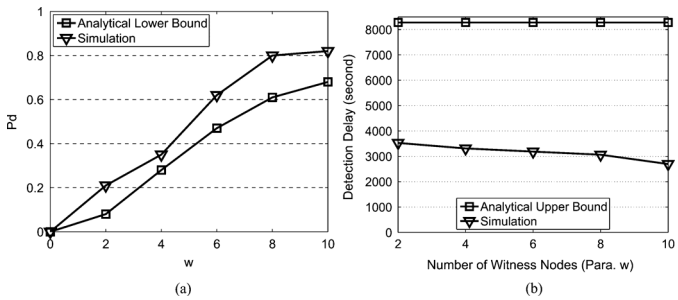


Fig. 10. Comparison of analysis and simulation results on the detection probability and detection delay of a single misreporting instance. The synthetic trace is used. (a) Detection probability. (b) Detection delay.

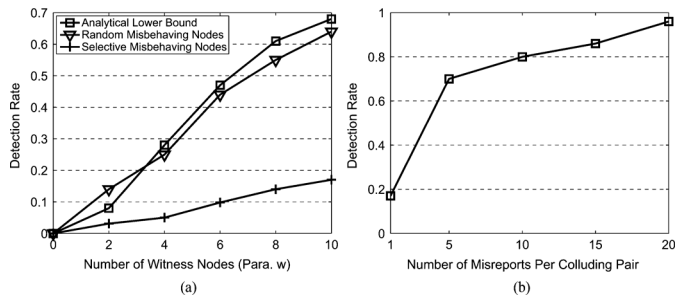


Fig. 11. Detection rate of our scheme in the Reality trace. (a) Each collusion pair misreports once. (b) Parameter $w = 10$.

approximation. The detection delay in the simulations is lower than the analytical upper bound.[3]

Then we evaluate the detection rate of our scheme on the Reality trace. As shown in Fig. 11(a), the detection rate increases as the number of witness nodes for each record summary increases. When misbehaving nodes are randomly deployed, the detection rate is close to the analytical result, which implies that the set of witness nodes randomly selected from a node's local view can be roughly seen as a random subset of the global node set. However, when misbehaving nodes are selectively deployed, the detection rate is much lower since misbehaving nodes contact normal nodes less frequently and a forged record has less chances to be disseminated by a normal node. Despite this, our scheme can still effectively detect the selectively deployed misbehaving nodes when they launch more misreporting instances. As shown in Fig. 11(b), when each colluding pair misreports 20 times, 96% of them are detected.

Next we evaluate the detection delay of our scheme on the Reality trace. To better evaluate the delay caused by node mobility, we set T_{delete} as infinity in this group of simulations. Fig. 12 shows the cdf of detection delay compared with packet delivery delay when Delegation is used in the Reality trace. It shows that 80% of misreporting instances are detected within 20 days, but it needs 40 days to deliver 80% of the packets. Thus, the detection delay is much shorter than the packet delivery delay.

3) Cost: The size of record and summary is set as follows: node ID, sequence number, and timestamp have 4 B each; a hash has 16 B; a signature has 40 B [30]. In default, $w = 4$ and $T_{\text{delete}} = 30$ days. Delegation is used and all nodes are normal.

[3]In the Random Waypoint model, $\lambda \approx 8\omega rv/\pi S$[26], where $\omega \approx 1.3683$, r is the transmission range of node, v is the moving speed of node (we use the average moving speed 1.3), and S is the size of the simulation area.

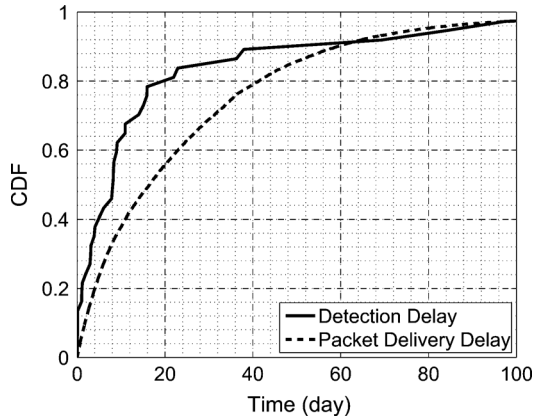


Fig. 12. Detection delay compared with the packet delivery delay.

TABLE I
AVERAGE COMMUNICATION OVERHEAD PER CONTACT

w	2	4	6	8	10
Communication (KB)	10.3	10.5	10.8	11.3	11.5
Pkt. Generation Rate (pkt/node/day)	0.5	1	2	4	8
Communication (KB)	6.7	10.5	14.7	20.6	27.5

TABLE II
AVERAGE STORAGE OVERHEAD PER NODE

w	2	4	6	8	10
Storage (KB)	47	71	89	108	127
T_{delete} (days)	20	30	40	50	60
Storage (KB)	45	71	92	126	170
Pkt. Generation Rate (pkt/node/day)	0.5	1	2	4	8
Storage (KB)	70	71	72	74	79

The communication cost of our scheme is given in Table I. We can see that the communication overhead increases with the parameter w, but very slowly. This is because w only affects how many times a record summary is transmitted. Since only one record summary is generated per contact, the transmission of summaries is only a minor source of communication. In contrast, the major source of communication overhead comes from the reporting of contact records which includes the vector of buffered packets. For this reason, when the packet generation rate increases, the communication overhead increases significantly as shown in Table I. However, the overall communication overhead is still low, e.g., less than 30 KB when each node generates 10 packets per day.

The storage cost of our scheme is shown in Table II. The storage overhead increases significantly with the parameter w and T_{delete} since record summaries are stored at more nodes and for a longer time. However, the storage overhead only increases slowly with the packet generation rate. This is because the major source of storage overhead is record summaries which are stored for a relatively long time, not contact records which are deleted soon, and the number of generated record summaries only depends on the number of contacts, not on the traffic load. Generally, the storage overhead of our scheme is low, less than 200 KB at each node.

IX. CONCLUSION

In this paper, we presented a scheme to detect packet dropping in DTNs. The detection scheme works in a distributed way;

i.e., each node detects packet dropping locally based on the collected information. Moreover, the detection scheme can effectively detect misreporting even when some nodes collude. Analytical results on detection probability and detection delay were also presented. Based on our packet dropping detection scheme, we then proposed a scheme to mitigate routing misbehavior in DTNs. The proposed scheme is very generic and it does not rely on any specific routing algorithm. Trace-driven simulations show that our solutions are efficient and can effectively mitigate routing misbehavior.

REFERENCES

[1] K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proc. SIGCOMM*, 2003, pp. 27–34.

[2] J. Burgess, B. Gallagher, D. Jensen, and B. Levine, "Maxprop: Routing for vehicle-based disruption-tolerant networks," in *Proc. IEEE INFOCOM*, 2006, pp. 1–11.

[3] W. Gao and G. Cao, "User-centric data dissemination in disruption tolerant networks," in *Proc. IEEE INFOCOM*, 2011, pp. 3119–3127.

[4] E. Daly and M. Haahr, "Social network analysis for routing in disconnected delay-tolerant manets," in *Proc. ACM MobiHoc*, 2007, pp. 32–40.

[5] V. Erramilli, A. Chaintreau, M. Crovella, and C. Diot, "Delegation forwarding," in *Proc. ACM MobiHoc*, 2008, pp. 251–260.

[6] N. Eagle and A. Pentland, "Reality mining: Sensing complex social systems," *Pers. Ubiquitous Comput.*, vol. 10, no. 4, pp. 255–268, 2006.

[7] J. Burgess, G. D. Bissias, M. Corner, and B. N. Levine, "Surviving attacks on disruption-tolerant networks without authentication," in *Proc. ACM MobiHoc*, 2007, pp. 61–70.

[8] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *Proc. ACM MobiCom*, 2000, pp. 255–265.

[9] H. Yang, J. Shu, X. Meng, and S. Lu, "Scan: Self-organized network-layer security in mobile ad hoc networks," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 2, pp. 261–273, 2006.

[10] S. Buchegger and Y. L. Boudec, "Performance analysis of the confidant protocol (cooperation of nodes: Fairness in dynamic ad-hoc networks)," in *Proc. MobiHoc*, 2002, pp. 226–236.

[11] K. Liu, J. Deng, P. K. Varshney, and K. Balakrishnan, "An acknowledgment-based approach for the detection of routing misbehavior in MANETs," *IEEE Trans. Mobile Comput.*, vol. 6, no. 5, pp. 536–550, May 2007.

[12] B. Awerbuch, D. Holmer, C.-N. Rotaru, and H. Rubens, "An on-demand secure routing protocol resilient to byzantine failures," in *Proc. ACM WiSe*, 2002, pp. 21–30.

[13] Y. Xue and K. Nahrstedt, "Providing fault-tolerant ad-hoc routing service in adversarial environments," *Wireless Pers. Commun.*, vol. 29, no. 3-4, pp. 367–388, 2004.

[14] P. Hui, J. Crowcroft, and E. Yoneki, "Bubble rap: Social-based forwarding in delay tolerant networks," in *Proc. ACM MobiHoc*, 2008, pp. 241–250.

[15] F. Li, A. Srinivasan, and J. Wu, "Thwarting blackhole attacks in disruption-tolerant networks using encounter tickets," in *Proc. IEEE INFOCOM*, 2009, pp. 2428–2436.

[16] Y. Ren, M. C. Chuah, J. Yang, and Y. Chen, "Detecting wormhole attacks in delay tolerant networks," *IEEE Wireless Commun. Mag.*, vol. 17, no. 5, pp. 36–42, Oct. 2010.

[17] U. Shevade, H. Song, L. Qiu, and Y. Zhang, "Incentive-aware routing in dtns," in *Proc. IEEE ICNP*, 2008, pp. 238–247.

[18] B. Chen and C. Choon, "Mobicent: A credit-based incentive system for disruption tolerant network," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[19] Q. Li, S. Zhu, and G. Cao, "Routing in socially selfish delay tolerant networks," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[20] Q. Li, W. Gao, S. Zhu, and G. Cao, "A routing protocol for socially selfish delay tolerant networks," in *Ad Hoc Networks*, Aug. 2011, DOI: 10.1016/j.adhoc.2011.07.007.

[21] B. Parno, A. Perrig, and V. Gligor, "Distributed detection of node replication attacks in sensor networks," in *IEEE Symp. Security and Privacy*, 2005, pp. 49–63.

[22] W. Gao and G. Cao, "On exploiting transient contact patterns for data forwarding in delay tolerant networks," in *Proc. IEEE ICNP*, 2010, pp. 193–202.

[23] C. Gentry and A. Silverberg, "Hierarchical id-based cryptography," in *Proc. Int. Conf. Theory and Application of Cryptography and Information Security*, 2002, pp. 548–566.

[24] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav, "Lowcost communication for rural internet kiosks using mechanical backhaul," in *ACM Proc. Mobicom*, 2006, pp. 334–345.

[25] A. Vahdat and D. Becker, Epidemic routing for partially connected ad hoc networks Duke University, Tech. Rep. CS-200006, 2000.

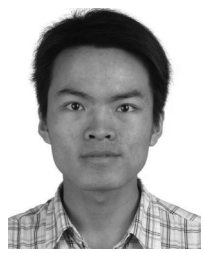
[26] R. Groenevelt, Stochastic Models in Mobile ad hoc Networks University of Nice, Sophia Antipolis, INRIA, 2006.

[27] V. Conan, J. Leguay, and T. Friedman, "Characterizing pairwise intercontact patterns in delay tolerant networks," *ACM Autonomics*, pp. 19:1–19:9, 2007.

[28] W. Gao, Q. Li, B. Zhao, and G. Cao, "Multicasting in delay tolerant networks: A social network perspective," in *Proc. ACM MobiHoc*, 2009, pp. 299–308.

[29] H. Zhu, L. Fu, G. Xue, Y. Zhu, M. Li, and L. Ni, "Recognizing exponential inter-contact time in vanets," in *Proc. IEEE INFOCOM*, 2010, pp. 101–105.

[30] J. C. Cha and J. H. Cheon, "An identity-based signature form gap diffie-hellman groups," in *Proc. PKC*, 2003, pp. 18–30.



Qinghua Li (S'09) received the B.E. degree from Xian Jiaotong University, China, and the M.S. degree from Tsinghua University, China, in 2004 and 2007, respectively. He is currently working toward the Ph.D. degree in the Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA.

His research interests include wireless networks and network security.

Guohong Cao (S'98–A'99–M'03–SM'06–F'11) received the B.S. degree from Xian Jiaotong University, China. He received the M.S. and Ph.D. degrees in computer science from the Ohio State University, in 1997 and 1999, respectively.

Since then, he has been with the Department of Computer Science and Engineering at the Pennsylvania State University, University Park, PA, where he is currently a Professor. His research interests are wireless networks and mobile computing. He has published more than 150 papers in the areas of wireless network security, wireless sensor networks, vehicular ad hoc networks, cache management, data access and dissemination, and distributed fault tolerant computing. He has served on the editorial board of IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, and has served on the organizing and technical program committees of many conferences.

Dr. Cao was a recipient of the NSF CAREER award in 2001.